Apple Assembly Line

Volume 1 -- Issue 10 July, 1981

In This Issue...

Renewing Subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for.

If your label says "8109", now is the time to renew to be sure of uninterrupted service.

Beneath Apple DOS

In the few weeks since I sent out last month's AAL, with the review of this book, I have sold 85 copies! My apologies if your shipment was delayed a little. Last Friday at 3:30 a shipment of 100 copies arrived; at 5:45 I took about 50 packages to the UPS station. Another 10 went out by mail this morning. A lot of work, but a lot of fun too.

I expect another shipment of 100 copies about the time you get this newsletter, so go ahead and order your copy if you have been waiting.

Using Firmware Card in Slot 4

Are you tired of getting "LANGUAGE NOT AVAILABLE" errors? Do you have a 16K RAM card, and also an old Firmware Card with one of the Basics on it? You can patch DOS to allow the Firmware Card to be put in slot 4, and still keep your RAM card in slot 0 for Pascal or whatever. With DOS loaded, type CALL -151 to get to the monitor; then patch:

```
     *A5B8:C0
     *A5C0:C1
```

Get back into Basic (3D0G), and INIT a disk with the modified DOS. If you have a disk utility program, you can patch the DOS image on an existing disk the same way. (From Michael W. Sanders, Decatur, GA)

The Lower Case Apple.........................Bob Matzinger

It occured to me that, since I have installed a Dan Paymar Lower Case Adapter, there ought to be a better way to generate lower case characters than by RAM-resident software.

The major problem is the F8 ROM. The CAPTST routine at $FD7E will not allow lower case characters to pass; if they get this far, they will be converted to upper case here. I cannot figure a reason for this routine, since the Apple will not generate lower case codes in the first place!

Anyway, there are only two ways I know of to avoid CAPTST: write my own line input subroutine (I want to avoid that!), or burn a new F8 ROM. All I would have to change is one lousy byte, at $FD83, from $DF to $FF. Seems like a waste of time...or is it? Maybe, since I am going to the trouble of burning the ROM, I can add some routines to extend the capabilities of my keyboard to access ALL of the ASCII characters.

That is what I decided to do. But! How do I make it transparent? It should not interfere with or be interfered by any program or language.

Within the monitor routines there are two that are not used; in fact, they were removed when the Autostart ROM came about. These are the 16-bit multiply and divide routines from $FB60 through $FBC0. I can insert my new code there.

I also need two RAM locations for shift lock and case flags. I must find two locations that would probably NOT be used by any other program. There are a number of location in zero page that are not normally used; the bottom of the stack and the top of the input buffer might not be used. Checking that out, however, I have found that most other people have thought of these locations already. Where can I go?

I found two bytes not used by anyone, inside the screen buffer area. They are reserved for the board plugged into slot 6, which in my case is the disk controller. The disk controller does not use locations $077E and $07FE ($0778+slot# and $07F8+slot#). More than likely, nobody would use these locations (at least that is what I am gambling on).

Now that I have room for flags, the next step is to write the routines to fit between $FB60 and $FBC0, and set up calls to them. I have to be careful not to change any other routines. Here is what I want:

1. Upon RESET, initialize to upper case.
2. Have a shift and shift-lock routine.
3. Be able to enter all ASCII characters.

When RESET is pressed, or when the Apple is turned on, the 6502 microprocessor executes a JMP indirect using the address at $FFFC and $FFFD. This effectively jumps to $FF59 in the monitor which is the reset routine. The reset routine calls INIT at $FB2F, which in turn ends with a JMP VTAB at $FB5D. If I change that last instruction, it can fall into the area formerly occupied by the multiply routine. How convenient! I'll put the code there to set upper case mode.

Most programs written for use with the Paymar Adapter have their own input routines. The monitor routines are not used. Therefore my changes should have no adverse effect on these programs.

The next thing I had to decide was which control-keys to use for shift, shift-lock, and the three characters not available from the standard Apple keyboard. I didn't want to use the escape key, since it is used by so many other programs. I finally chose these:

control-Z:	Shift and Shift-lock
control-K:	Left bracket and Left Brace
control-L:	Backslash and Vertical Bar
control-O:	Underline and Rubout

One final problem to overcome is passing the cursor over a lower case character. The cursor, in the normal monitor, makes the character under the cursor flash. A lower case character will flash in upper case, so you cannot tell whether it was lower or upper case without moving the cursor. I decided to make lower case characters under the cursor display as inverse upper case, rather than flashing. That way there is no doubt.

Now how do we get the patches into the ROM? First we need to get a copy of the standard ROM code into RAM. Then assemble the patches, and save the patched copy on disk. From inside the S-C Assembler II, type:

```
     :$6800<F800.FFFFM               (copy monitor into RAM)
     :ASM                            (assemble the patches)
     :BSAVE F8 EPROM,A$6800,L$800    (save patched monitor)
```

After the patches had been made, I used ROMWRITER, by Mountain Hardware, to burn a 2716 EPROM. This EPROM was then inserted, with appropriate adaptation, in the F8 socket on my Apple mother board.

[NOTE: A 2716 EPROM WILL NOT DIRECTLY REPLACE THE F8 ROM. EITHER THE MOTHER BOARD CIRCUITRY MUST BE MODIFIED OR AN APPROPRIATE SOCKET ADAPTER MUST BE USED.]

If you have a 16K RAM card, you can try the patched monitor without burning a ROM. After the patches have been assembled into the standard copy at $6800, type the following:

```
     :$C081 C081                  (write enable RAM card)
     :$F800<6800.6FFFM            (copy new monitor up)
     :$C080                       (turn on RAM version)
```

After putting the patched monitor into the RAM card, you have to patch the assembler to turn off its own CAPTST, if you want to see the lower case stuff work inside the assembler. Type:

```
:$139B:FF
```

This will make the assembler allow lower case characters to be typed in, but they are only legal in comments.

Some more words of caution. These patches are for the "old" monitor ROM. They will not work in the Autostart ROM. My choice of control-K and control-L may upset some users. Control-K is used as a monitor command equivalent for IN#slot, and control-L is used to generate a form-feed on some printers. I can always go to BASIC for the IN#slot, and my printer has a button for form-feed. I feel that the full upper-lower case ability is much more desirable.

WHEN ALL ELSE FAILS, READ THE INSTRUCTIONS AGAIN!

```
                  1000 * LOWER CASE F8 ROM.1
                  1010 *-------------------------------
                  1020 * THESE PATCHES ARE FOR THE "OLD" F8 ROM.  THEY
                  1030 * WILL NOT WORK INTO THE AUTOSTART ROM MONITOR
                  1040 * ROUTINES.
                  1050 *
                  1060 * OPERATION: $6800<F800.FFFFM
                  1070 *            ASM  (ASSEMBLE THIS CODE)
                  1080 *            BSAVE F8 EPROM,A$6800,L$0800
                  1090 *-------------------------------
008B-             1100 CTRLK  .EQ $8B        LEFT BRACKET OR BRACE
008C-             1110 CTRLL  .EQ $8C        BACKSLASH OR VERTICAL BAR
008F-             1120 CTRLO  .EQ $8F        UNDERLINE OR RUBOUT
009A-             1130 CTRLZ  .EQ $9A        SHIFT OR SHIFT LOCK
077E-             1140 CASE   .EQ $77E       FOR DOS IN SLOT 6
07FE-             1150 LCKFLG .EQ $7FE       FOR DOS IN SLOT 6
C010-             1160 KYSTRB .EQ $C010
FC22-             1170 VTAB   .EQ $FC22
FD0C-             1180 RDKEY  .EQ $FD0C
                  1190 *-------------------------------
                  1200 PATCH1 .OR $FB5D
                  1210        .TA $6B5D
                  1220 *
FB5D- A0 00       1230 SETCAS LDY #0         PART OF RESET ROUTINE TO INIT
FB5F- 8C 7E 07    1240        STY CASE       UPPER CASE MODE
FB62- C8          1250        INY
FB63- 8C FE 07    1260        STY LCKFLG
FB66- 4C 22 FC    1270        JMP VTAB
                  1280 *-------------------------------
                  1290 PATCH2 .OR $FD2B
                  1300        .TA $6D2B
                  1310 *
FD2B- 4C 69 FB    1320        JMP LCADAP     FROM KEYIN ROUTINE TO LOWER
FD2E- EA          1330        NOP            CASE "ADAPTER"
                  1340 *-------------------------------
                  1350 PATCH3 .OR $FD82
                  1360        .TA $6D82
                  1370 *
FD82- 29 FF       1380        AND #$FF       ALLOW LOWER CASE TO PASS
                  1390 *-------------------------------
                  1400 PATCH4 .OR $FD11
                  1410        .TA $6D11
                  1420 *
FD11- 20 B5 FB    1430        JSR FORM       DISPLAY CHARACTERS UNDER THE
FD14- EA          1440        NOP            CURSOR CORRECTLY
                  1450 *-------------------------------
                  1460 * THE CTRL-Z KEY IS USED LIKE THE SHIFT KEY ON A
                  1470 * TYPEWRITER:  ONE CTRL-Z WILL ENTER ONE UPPER
                  1480 * CASE CHARACTER AND THEN RETURN TO LOWER CASE.
                  1490 *
                  1500 * TWO CTRL-Z'S IN SUCCESSION WILL PERFORM A
                  1510 * "SHIFT-LOCK".  IF THE MODE WAS LOWER CASE,
                  1520 * TWO CTRL-Z'S WILL LOCK IN UPPER CASE; IF THE
                  1530 * MODE WAS UPPER CASE, TWO CTRL-Z'S WILL LOCK
                  1540 * IN LOWER CASE.
                  1550 *-----------------------------------
                  1560 PATCH5 .OR $FB69
                  1570        .TA $6B69
                  1580 *
FB69- 2C 10 C0    1590 LCADAP BIT KYSTRB      CLEAR KEYBOARD
FB6C- C9 9A       1600        CMP #CTRLZ      SEE IF "SHIFT"
FB6E- D0 1A       1610        BNE .4          NO, TRY OTHER TESTS
FB70- AD FE 07    1620        LDA LCKFLG
FB73- 49 80       1630        EOR #$80        FLIP BIT 7 (CTRLZ FLAG)
FB75- 30 02       1640        BMI .1          NEGATIVE IF FIRST CTRL-Z
FB77- 49 01       1650        EOR #$01        FLIP BIT 0 (LOCK FLAG)
FB79- 8D FE 07    1660 .1     STA LCKFLG
FB7C- F0 04       1670        BEQ .2          ...IF LOCK FLAG IS CLEAR
FB7E- A9 00       1680        LDA #0          SET UPPER CASE
FB80- F0 02       1690        BEQ .3          ...ALWAYS
FB82- A9 20       1700 .2     LDA #$20        SET LOWER CASE
FB84- 8D 7E 07    1710 .3     STA CASE
FB87- 4C 0C FD    1720        JMP RDKEY
FB8A- C9 8B       1730 .4     CMP #CTRLK
FB8C- F0 08       1740        BEQ .5
FB8E- C9 8C       1750        CMP #CTRLL
FB90- F0 04       1760        BEQ .5
FB92- C9 8F       1770        CMP #CTRLO
FB94- D0 02       1780        BNE .6
FB96- 09 50       1790 .5     ORA #$50        CONVERT TO SPECIAL CHARS
FB98- C9 C0       1800 .6     CMP #$C0        MERGE CASE IF ALPHA
FB9A- 90 03       1810        BCC .7          NOT ALPHA
FB9C- 0D 7E 07    1820        ORA CASE
FB9F- 48          1830 .7     PHA             SAVE MODIFIED CHAR
FBA0- AD FE 07    1840        LDA LCKFLG
FBA3- 10 05       1850        BPL .8          ...IF Z-FLAG CLEAR
FBA5- A9 00       1860        LDA #0          CLEAR Z AND LOCK FLAGS
FBA7- 8D FE 07    1870        STA LCKFLG
FBAA- D0 05       1880 .8     BNE .9          ...IF LOCK FLAG IS SET
FBAC- A9 20       1890        LDA #$20        SET LOWER CASE
FBAE- 8D 7E 07    1900        STA CASE
FBB1- 68          1910 .9     PLA             RETRIEVE MODIFIED CHAR
FBB2- 60          1920        RTS
FBB3- 00          1930        BRK
FBB4- 00          1940        BRK
                  1950 *-----------------------------------
                  1960 * CURSOR DISPLAY FOR EDITING
                  1970 *
FBB5- C9 E0       1980 FORM   CMP #$E0        IS IT LOWER CASE?
FBB7- B0 05       1990        BCS .1          YES, SO BRANCH
FBB9- 29 3F       2000        AND #$3F        ALL CHARACTERS (EXCEPT LOWER
FBBB- 09 40       2010        ORA #$40        CASE) ARE FLASHED
FBBD- 60          2020        RTS
FBBE- 49 E0       2030 .1     EOR #$E0        MAKE LOWER CASE INTO
FBC0- 60          2040        RTS              INVERSE UPPER CASE
                  2050 *-----------------------------------
                  2055 * WRITTEN:  NOVEMBER 1, 1980
                  2060 * REVISED:  JUNE 25, 1981
                  2070 *  AUTHOR:  BOB MATZINGER
                  2080 *           P. O. BOX 13446
                  2090 *           ARLINGTON, TX 76013
                  2100 *            (817) 265-8122
                  2110 *-----------------------------------
```

Screen Printer

Last month I alluded to my trouble in getting a screen printing subroutine to work with the Apple Parallel Interface. I finally got it going, and now it doesn't look hard at all.

The program is set up to be loaded and started with a BRUN command. This doesn't start any printing, however. The initial code just puts a hook address into location $38 and $39, and passes them to DOS. Thereafter, all character-input calls will have to go through my routine at lines 1260-1320 (SCRN.PRNT).

The SCRN.PRNT subroutine looks at each input character to see if it is a control-P (ASCII code = $90). If not, the character is passed on to whatever program tried to read a character. If it is a control-P, the current contents of the screen are printed.

(My printer is in slot 1; if you are using a different slot, change lines 1110 and 1120.)

The actual printing subroutine is really straightforward. It consists of four parts: 1) save current registers and cursor position; 2) initialize Apple Parallel Interface temporaries; 3) print each line of the screen on the printer; and 4) restore the cursor position and registers.

Lines 1350-1410 save the A-, X-, and Y-registers on the stack, followed by the cursor horizontal position. I pushed them on the stack rather than allocate temporaries, but either way will work. Using the stack saves a few bytes of code and 4 bytes of temporary memory, but it takes a few more cycles if you are worried about speed.

Lines 1420-1490 initialize the temporaries used by the code in Apple's Parallel Interface ROM. These temporaries are actually inside the screen buffer memory (between $0400 and $07FF), but they are in bytes that do not get displayed. (There are 64 bytes in the screen buffer that do not get displayed, and which are used by interface cards for temporary memory. These are $478-47F, $4F8-4FF, $578-57F, $5F8-5FF, $678-67F, $6F8-6FF, $778-77F, and $7F8-7FF.) For more information on how the Parallel Interface uses these temporaries, see your manual.

Lines 1500-1670 actually print the screen contents. The X-register is used as a line counter, and runs from 0 to 23. See lines 1500, 1510, and 1650-1670. This is quite analogous to a BASIC statement like FOR I=0 TO 23.

Inside the X-loop, line 1520 computes a new base address for the current line. Then the Y-register is used as a column counter. Lines 1530 and 1600-1620 control the Y-loop. Inside the Y-loop, each character of the line is picked up in turn. Lines 1550-1580 convert inverse or flashing characters to normal ASCII codes for printing. Line 1590 calls on the Parallel Interface program to print one character. (The entry at $Cx02 assumes all temporaries are already set up.) At the end of each line, lines 1630 and 1640 send a carriage return to the printer.

Lines 1680-1700 restore the cursor position and base address pointer, and lines 1710-1750 restore the 6502 registers.

I wrote this program, lines 1340-1760, as a subroutine even though it could have been in-line. I did it so that you can call it directly from your Applesoft or Integer BASIC program, with a "CALL 793". This feature makes the very-valuable screen printer even more useful.

```
                  1000 *--------------------------------
                  1010 *      SCREEN PRINTER
                  1020 *--------------------------------
0024-             1030 MON.CH      .EQ $24
0028-             1040 MON.BASL    .EQ $28,29
FBC1-             1050 MON.BASCAL  .EQ $FBC1
FC22-             1060 MON.VTAB    .EQ $FC22
FD0C-             1070 MON.RDKEY   .EQ $FD0C
FD1B-             1080 MON.KEYIN   .EQ $FD1B
03EA-             1090 DOS.REHOOK  .EQ $3EA
                  1100 *--------------------------------
0001-             1110 SLOT   .EQ 1
C102-             1120 PRINT  .EQ $C102     $C002+SLOT*256
05F9-             1130 MSTRT  .EQ $5F8+SLOT
0679-             1140 MODE   .EQ $678+SLOT
06F9-             1150 ESCHAR .EQ $6F8+SLOT
0779-             1160 FLAGS  .EQ $778+SLOT
                  1170 *--------------------------------
                  1180        .OR $300
                  1190 *--------------------------------
0300- A9 0B       1200        LDA #SCRN.PRNT
0302- 85 38       1210        STA $38
0304- A9 03       1220        LDA /SCRN.PRNT
0306- 85 39       1230        STA $39
0308- 4C EA 03    1240        JMP DOS.REHOOK
                  1250 *--------------------------------
                  1260 SCRN.PRNT
030B- 20 1B FD    1270        JSR MON.KEYIN  GET CHAR
030E- C9 90       1280        CMP #$90       CONTROL-P?
0310- D0 06       1290        BNE .1
0312- 20 19 03    1300        JSR SCREEN.PRINTER
0315- 4C 0C FD    1310        JMP MON.RDKEY
0318- 60          1320 .1     RTS
                  1330 *--------------------------------
                  1340 SCREEN.PRINTER
0319- 48          1350        PHA            SAVE REGS
031A- 8A          1360        TXA
031B- 48          1370        PHA
031C- 98          1380        TYA
031D- 48          1390        PHA
031E- A5 24       1400        LDA MON.CH     SAVE CH
0320- 48          1410        PHA
0321- A9 28       1420        LDA #40        SET UP APPLE CONTROLLER ROM
0323- 8D F9 05    1430        STA MSTRT      TEMPORARIES
0326- A9 00       1440        LDA #0
0328- 8D 79 06    1450        STA MODE
032B- A9 89       1460        LDA #$89
032D- 8D F9 06    1470        STA ESCHAR
0330- A9 01       1480        LDA #1
0332- 8D 79 07    1490        STA FLAGS
0335- A2 00       1500        LDX #0         START AT LINE 0
0337- 8A          1510 .1     TXA
0338- 20 C1 FB    1520        JSR MON.BASCAL  COMPUTE BASE POINTER FOR LINE
033B- A0 00       1530        LDY #0         START AT CHAR 0
033D- B1 28       1540 .2     LDA (MON.BASL),Y
033F- C9 A0       1550 .3     CMP #$A0       MAP FLASH AND INVERSE TO NORMAL
0341- B0 04       1560        BCS .4
0343- 69 40       1570        ADC #$40
0345- D0 F8       1580        BNE .3         ...ALWAYS
0347- 20 02 C1    1590 .4     JSR PRINT
034A- C8          1600        INY            NEXT CHARACTER
034B- C0 28       1610        CPY #40        END OF LINE?
034D- 90 EE       1620        BCC .2         NO
034F- A9 8D       1630        LDA #$8D       YES, PRINT CARRIAGE RETURN
0351- 20 02 C1    1640        JSR PRINT
0354- E8          1650        INX            NEXT LINE
0355- E0 18       1660        CPX #24        END OF SCREEN
0357- 90 DE       1670        BCC .1         NO
0359- 68          1680        PLA            YES, RESTORE CH
035A- 85 24       1690        STA MON.CH
035C- 20 22 FC    1700        JSR MON.VTAB   RESTORE BASE POINTER
035F- 68          1710        PLA            RESTORE REGS
0360- A8          1720        TAY
0361- 68          1730        PLA
0362- AA          1740        TAX
0363- 68          1750        PLA
0364- 60          1760        RTS
```

Restoring Clobbered Page 3 Pointers........Preston R. Black, M.D.

Here's a very short (14 byte) program which you might find useful. As you know, DOS writes the page 3 vectors (between $3D0 and $3FF) as the last step in the bootstrap process. This is done by copying a portion of DOS onto this area. The image remains in memory and can be used to rewrite the vectors if they are clobbered.

If you have a 48K Apple, the routine which copies the vector data starts at $9E25. My program temporarily patches DOS to isolate the vector-copier, by storing an RTS opcode at the end of the loop ($9E30). After calling the loop, the original value of $9E30 is restored.

I put the subroutine at $BCD0 inside DOS, abecause this area is not used by DOS. It can be placed on all slave diskettes you INIT after patching DOS. With this subroutine installed, you can use all of page 3 for your assembly language program. Once your program is finished, you can JMP $BCD0 to restore $3D0-$3FF to its normal state.

Here is the program, written to assemble into $0CD0-0CDD. After assembly is complete, you can move it into DOS with the monitor command

```
     :$BCD0<CD0.CDDM    (if issued from inside S-C Assembler II
     or
     *BCD0<CD0.CDD      (if you do it from the monitor.
```

```
                  1000 *---------------------------------
                  1010 *      RESTORE PAGE 3 VECTORS
                  1020 *      ----------------------
                  1030 *
                  1040 *      PRESTON R. BLACK, M.D.
                  1050 *      12 JUNE 1981
                  1060 *---------------------------------
                  1070        .OR $BCD0
                  1080        .TA $0CD0
                  1090 *---------------------------------
                  1100 RESTORE.PAGE.3.VECTORS
BCD0- A9 60       1110        LDA #$60     RTS OPCODE
BCD2- 8D 30 9E    1120        STA $9E30
BCD5- 20 25 9E    1130        JSR $9E25
BCD8- A9 AD       1140        LDA #$AD     ORIGINAL DATA
BCDA- 8D 30 9E    1150        STA $9E30
BCDD- 60          1160        RTS
```

On second thought, 12 bytes is enough. Rather than patching the DOS code to make a subroutine, I can just put a program up at $BCD0 which looks like the code at $9E25. Here is the shorter version:

```
                  1070        .OR $BCD0
                  1080        .TA $0CD0
                  1090 *---------------------------------
                  1100 RESTORE.PAGE.3.VECTORS
BCD0- A2 2F       1110        LDX #$3FF-$3D0 # BYTES TO BE COPIED
BCD2- BD 51 9E    1120 .1     LDA $9E51,X    ADDRESS OF VECTORS INSIDE DOS
BCD5- 9D D0 03    1130        STA $3D0,X     VECTOR AREA
BCD8- CA          1140        DEX
BCD9- 10 F7       1150        BPL .1
BCDB- 60          1160        RTS
```

Corrections to Variable Cross Reference Program

The Variable Cross Reference program I printed in issue #2 (November, 1980) had at least three bugs. One of them was reported a long time ago, but I had no idea what the cause was until today. The other two were never reported by anyone, but I discovered their presence and cause today. Eventful day!

Bug #1: After using the VCR program, the first line number LISTed by a subsequent LIST command printed out with all sorts of extra fractional digits. Strange! I finally tracked it down to a page zero location which VCR used. Location $A4 is left with a non-zero value, but Applesoft expects and requires it to be zero. If it is not zero, the floating point multiply subroutine gives wrong answers. The multiplication failure ruins the first number printed after running VCR.

Solution to Bug #1: Add the following two lines to the VCR program.

```
     1452          LDA #0       CLEAR $A4 FOR APPLESOFT
     1454          STA $A4
```

Bug #2: The logic for terminating the main program loop (lines 1400-1460) was wrong, and resulted in sometimes adding a phony variable.

Solution to Bug #2: Delete line 1810, and change or add the following lines.

```
     1650          LDY #3       CAPTURE POINTER AND LINE #
     1692          LDA DATA+1   TEST FOR END
     1694          BEQ .3       YES
     1820 .3       RTS
```

Bug #3: If your program contained a PRINT statement with a quoted string not separated from a variable by a semi-colon or comma, the GET.NEXT.VARIABLE subroutine would invent new variable names from inside the quoted string! For example, the line PRINT D$"OPEN FILE" would add variables OP (for OPEN) and FI (for FILE).

Solution to Bug #3: Change or add the following lines.

```
     2752          BEQ .6       YES
     2754          CMP #'"      QUOTATION MARK?
     2762          LDA PNTR     BACK UP PNTR OVER QUOTE MARK
     2763          BNE .7
     2764          DEC PNTR+1
     2765 .7       DEC PNTR
     2766          RTS
     2770 .6       LDA VARNAM+2 SET HIGH BIT
```

If you have typed in the VCR program, or bought the Quarterly Disk #1 which contained the source, you should now go back and fix these three bugs. (All the line numbers above fit in with the program as printed last November.) Copies of the Quarterly Disk #1 with a serial number of 44 or higher already have been fixed.

Step-Trace Utility

The Motive:

"Not that it was that good, mind you! But we needed something, and they should not have yanked it out without providing some other way to debug machine language programs."

When Apple converted over to the Autostart ROM, they not only removed the hardly-ever-used 16-bit multiply and divide subroutines. They also stripped the S and T commands, which left assembly language programmers naked. How can you possibly debug complicated 6502 code without at least a single step capability?

Several programs are now on the market, in the $50 price range, which give you step, trace, breakpoints, stack display, et cetera. "John's Debugger", from John Broderick & Associates, 8635 Shagrock, Dallas, TX 75238 is one. Someone called me from Augusta, GA, yesterday to tell me about a similar package he has written and wants to market (I'll be reviewing this one; it may become an S-C SOFTWARE product). I saw another ad this month somewhere, but I cannot find it now.

But I wanted to do something special this month for the Assembly Line, so here is a limited STEP-TRACE program...free!

The Manner:

It is set up as a BRUNnable file, to load at $0800. If you want to load it somewhere else, you can put in an origin directive (.OR). The code executed when you BRUN the file (lines 1390-1460) merely installs the "control-Y vector". This enables the control-Y monitor command, which is a user-definable command.

Once the control-Y vector is loaded, you have two new commands. If you type a memory address and a control-Y (and a carriage return), the instruction at that memory address will be disassembled and displayed on line 23. The flashing cursor will be positioned at the end of the disassembled instruction. Just above the cursor, on line 22, you will see the current register contents. Line 24 is an inverse mode line which labels the registers, and reminds you of the options you have.

At this point you can type one of the five register names (A, X, Y, S, or P), or a space, or a carriage return. If you type a carriage return, the trace is aborted and you are returned to the assembler. If you type a space, the disassembled instruction will be exectuted. The new register contents will be displayed, the screen will scroll up, and the next instruction will be disassembled on line 23. If you type a register name, the cursor will be moved under that register. You can type in a new value for the register, and then hit a space for the next register or a return to get ready to execute again.

If you want to step through a little faster, hold down the space bar and the repeat key.

Once you have terminated the trace (by typing a carriage return), you can restart where you stopped by typing a control-Y and a carriage return. Since there is no address given, STEP-TRACE will begin where you stopped the last time. You can stop the trace, do some monitor commands, and then start tracing again.

Two warnings: I wrote STEP-TRACE to be used from inside the S-C ASSEMBLER II. That means all monitor commands, including the control-Y, need to be preceded by a dollar sign ($). If you want to use STEP-TRACE directly from the monitor, and not return inside the assembler after stopping, you need to change line 3500. It now says JMP $3D0, which restarts DOS and the assembler. Change it to JMP $FF69, which restarts the monitor. Line 3470 requires the .DA modification published in the December 1980 issue of AAL. If you haven't installed that yet, then rewrite line 3470 as five separate lines; if you don't, it will assemble without error but it will be WRONG!

The Method:

Now let's look through the listing, and see how it works. When the monitor decodes the control-Y command, the address you typed (if any) is loaded into $3C,3D in page zero. Then the monitor branches to $3F8, where we have already loaded a JMP STEP.TRACE instruction. We step into the action at line 1510.

Lines 1520-1570: the X-register is zero if no address was typed. In this case, we skip around the code to copy the address into MON.PC. If there was an address, copy it into MON.PC.

Lines 1580-1630: Set the stack pointer to $FF, giving the whole stack to the program under test. Move the cursor to the bottom of the screen and print a carriage return.

Lines 1650-1680: Call on subroutines to display the current register values (from the SAVE.AREA at line 4350-4400), disassemble the instruction pointed to by MON.PC, and wait on you to type something on the keyboard. This last subroutine does not return unless you type a space, indicating you want to execute the disassembled instruction.

Lines 1690-1860: Clear the XQT.AREA to NOP instructions. Get the stack pointer from the SAVE.AREA. Pick up the opcode byte, and see if it is one we have to interpret rather than execute (BRK, JSR, RTI, JMP, RTS, or JMP indirect). If so, jump to the appropriate code for each opcode.

Lines 1870-2010: Get the instruction length (less one) in Y, so we can copy the instruction into XQT.AREA. See if the opcode is one of the relative branches; if so, change the displacement to $04, so that we can execute it inside XQT.AREA. Copy the instruction bytes into XQT.AREA. Restore the registers from the SAVE.AREA, restoring status (P-register last of all.

Lines 2030-2160: Execute the instruction. Unless it is a relative branch instruction which branches, jump to did.not.branch. Relative branches which branch go to line 2100, where the effective address is computed and stored in MON.PC.

Lines 2180-2190: A BRK instruction displays the registers and returns to the assembler (aborts STEP-TRACE).

Lines 2210-2250: The RTI instruction checks the stack pointer; if there are not three bytes left on the stack, STEP-TRACE is aborted. If there are three left, the next byte is pulled off the stack and stored in the SAVE.AREA for the P-register. The rest of the RTI instruction is the same as an RTS istruction.

Lines 2260-2350: The RTS instruction checks the stack pointer; if there are not two bytes left on the stacke, STEP-TRACE is aborted. If there are two left, they are pulled off and stored in MON.PC.

Lines 2370-2470: The JSR instruction picks up the current MON.PC, adds two, and pushes the result on the stack. The new stack ponter value is saved in SAVE.AREA. Then a JMP instruction is simulated.

Lines 2480-2490: Simulate a JMP instruction by copying the address into MON.PC.

Lines 2500-2530: Simulate a JMP indirect instruction. Copy the address contained in the two bytes pointed to by the instruction address into MON.PC.

Lines 2550-2640: After a normal executed instruction, save all the registers in SAVE.AREA. Be sure the processor is in binary mode (not decimal).

Lines 2650-2690: Add the instruction length to MON.PC, and go back to get the next instruction.

Lines 2710-2800: Using the current MON.PC as a pointer, pick up the two bytes pointed to and put them into MON.PC. This is used by the JSR, JMP, and JMP indirect processors.

Lines 2820-2930: Set cursor position to line 23, column 27, and wait for you to type a key. If you type a carriage return, abort STEP-TRACE. If you type a space, return to whoever called WAIT.ON.KEYBOARD.

Lines 2940-2990: See if you typed a register name (letter A, X, Y, S, or P). If not, go back and wait till you type something else. If so, go on to line 3000.

Lines 3000-3100: Set inverse mode, position the cursor to the selected register column, and display the current contents of that register in inverse mode. Switch back to normal mode.

Lines 3110-3340: Wait again for you type a character on the keyboard. If you type a hexadecimal digit, shift the current register contents one digit position to the left, and add in the digit you just typed. (You can type as many digits as you want to; the last two you type will be the new contents.) If you type a space or a carriage return, branch to line 3350 or 3400.

Lines 3350-3390: You typed a space, so move over to the next register. If you just modified the S-register, move back to the A-register.

Lines 3400-3440: You typed a carriage return, so scroll up the screen and go back to the top of WAIT.ON.KEYBOARD.

Lines 3450-3470: REG.NAMES defines the register names. REG.INDEX is an index into REG.NAMES and REG.CH. REG.CH is a list of column positions for each of the registers. (If you have not installed the .DA modification from AAL Volume 1, Issue 3, you need to spread the data values out on five separate lines.)

Lines 3490-3500: Clear from the cursor to the end of screen, and return through DOS to the assembler. Change line 3500 if you want to go somewhere else after leaving the STEP-TRACE.

Lines 3540-3590: Adds the contents of the A-register to MON.PC.

Lines 3630-3740: Displays the register contents from SAVE.AREA.

Lines 3810-3840: Prints MON.PC and a dash. This is called by the disassembly subroutine.

Lines 3880-4330: Disassembles the instruction starting at MON.PC. This code is very similar to code in the Apple monitor ROM at $F882. It is modified slightly to change the spacing, so that there will be room for the register display on the same line.

```
                 1000 *---------------------------------
                 1010 *       STEP-TRACE UTILITY
                 1020 *---------------------------------
0023-            1030 MON.WNDBTM  .EQ $23
0024-            1040 MON.CH      .EQ $24
0025-            1050 MON.CV      .EQ $25
002C-            1060 LMNEM       .EQ $2C
002D-            1070 RMNEM       .EQ $2D
002E-            1080 MON.FORMAT  .EQ $2E
002F-            1090 MON.LENGTH  .EQ $2F
003A-            1100 MON.PC      .EQ $3A,3B
003C-            1110 MON.A1      .EQ $3C,3D
003E-            1120 MON.A2      .EQ $3E,3F
                 1130 *---------------------------------
03D0-            1140 DOS.REENTRY .EQ $3D0
03F8-            1150 Y.VECTOR    .EQ $3F8
07D0-            1160 BASE.LINE24 .EQ $7D0
F88E-            1170 MON.INSDS2  .EQ $F88E
F8D0-            1180 MON.INSTDSP .EQ $F8D0
F90C-            1190 MON.PRADDR  .EQ $F90C
F948-            1200 MON.PRBLNK  .EQ $F948
F94A-            1210 MON.PRBL2   .EQ $F94A
F9C0-            1220 MNEML       .EQ $F9C0
FA00-            1230 MNEMH       .EQ $FA00
FC22-            1240 MON.VTAB    .EQ $FC22
FC42-            1250 MON.CLREOP  .EQ $FC42
FC70-            1260 MON.SCROLL  .EQ $FC70
FC9C-            1270 MON.CLREOL  .EQ $FC9C
FD0C-            1280 MON.RDKEY   .EQ $FD0C
FD8E-            1290 MON.CROUT   .EQ $FD8E
FD99-            1300 MON.PRYX3   .EQ $FD99
FDDA-            1310 MON.PRBYTE  .EQ $FDDA
FDED-            1320 MON.COUT    .EQ $FDED
FE80-            1330 MON.SETINV  .EQ $FE80
FE84-            1340 MON.SETNORM .EQ $FE84
                 1350 *---------------------------------
C000-            1360 KEYBOARD   .EQ $C000
C010-            1370 STROBE     .EQ $C010
                 1380 *---------------------------------
                 1390 STEP.TRACE.SETUP
0800- A9 4C      1400        LDA #$4C      'JMP' OPCODE
0802- 8D F8 03   1410        STA Y.VECTOR
0805- A9 10      1420        LDA #STEP.TRACE
0807- 8D F9 03   1430        STA Y.VECTOR+1
080A- A9 08      1440        LDA /STEP.TRACE
080C- 8D FA 03   1450        STA Y.VECTOR+2
080F- 60         1460        RTS
                 1470 *---------------------------------
                 1480 *      (Y)           SINGLE STEP AT CURRENT PC
                 1490 *      ADR(Y)        SINGLE STEP AT ADR
                 1500 *---------------------------------
                 1510 STEP.TRACE
0810- 8A         1520        TXA           X=0 IF NO ADDRESSES
0811- F0 08      1530        BEQ .1        NO ADDRESSES
0813- A5 3C      1540        LDA MON.A1    ONE OR TWO ADDRESSES
0815- 85 3A      1550        STA MON.PC
0817- A5 3D      1560        LDA MON.A1+1
0819- 85 3B      1570        STA MON.PC+1
081B- A2 FF      1580 .1     LDX #$FF      USER GETS WHOLE STACK
081D- 9A         1590        TXS
081E- 8E 3C 0A   1600        STX SAVE.S
0821- A9 17      1610        LDA #23
0823- 85 25      1620        STA MON.CV
0825- 20 8E FD   1630        JSR MON.CROUT
                 1640 *---------------------------------
                 1650 TRACE.LOOP
0828- 20 97 09   1660        JSR DISPLAY.REGISTERS
082B- 20 DE 09   1670        JSR DISASSEMBLE ONE INSTRUCTION
082E- 20 F8 08   1680        JSR WAIT.ON.KEYBOARD
0831- A9 EA      1690        LDA #$EA      'NOP' OPCODE
0833- 8D 78 08   1700        STA XQT.AREA+1
0836- 8D 79 08   1710        STA XQT.AREA+2
0839- AE 3C 0A   1720        LDX SAVE.S
083C- 9A         1730        TXS
083D- A0 00      1740        LDY #0
083F- B1 3A      1750        LDA (MON.PC),Y  GET USER OPCODE
0841- F0 49      1760        BEQ X.BRK      'BRK' OPCODE
0843- C9 20      1770        CMP #$20       'JSR' OPCODE
0845- F0 66      1780        BEQ X.JSR
```

```
0847- C9 40     1790          CMP #$40      'RTI' OPCODE
0849- F0 47     1800          BEQ X.RTI
084B- C9 4C     1810          CMP #$4C      'JMP' OPCODE
084D- F0 6F     1820          BEQ X.JMP
084F- C9 60     1830          CMP #$60      'RTS' OPCODE
0851- F0 48     1840          BEQ X.RTS
0853- C9 6C     1850          CMP #$6C      'JMP ()' OPCODE
0855- F0 6D     1860          BEQ X.JMPI
0857- A4 2F     1870          LDY MON.LENGTH  # BYTES IN INSTRUCTION
0859- 29 1F     1880          AND #$1F      IF RELATIVE BRANCH, CHANGE
085B- 49 14     1890          EOR #$14      DISPLACEMENT TO $04
085D- C9 04     1900          CMP #$04      FOR XQT AREA
085F- F0 02     1910          BEQ .2
0861- B1 3A     1920 .1       LDA (MON.PC),Y  COPY INSTRUCTION INTO XQT AREA
0863- 99 77 08  1930 .2       STA XQT.AREA,Y
0866- 88        1940          DEY
0867- 10 F8     1950          BPL .1
0869- AD 3D 0A  1960          LDA SAVE.P    RESTORE ALL REGISTERS
086C- 48        1970          PHA
086D- AD 40 0A  1980          LDA SAVE.A
0870- AE 3F 0A  1990          LDX SAVE.X
0873- AC 3E 0A  2000          LDY SAVE.Y
0876- 28        2010          PLP
                2020 *--------------------------------
                2030 XQT.AREA
0877- EA        2040          NOP           USER'S OPCODE GOES HERE
0878- EA        2050          NOP
0879- EA        2060          NOP
087A- 4C CF 08  2070          JMP DID.NOT.BRANCH
                2080 *--------------------------------
                2090 *   RELATIVE BRANCHES THAT DO BRANCH COME HERE
087D- 18        2100          CLC
087E- A0 01     2110          LDY #1        GET ORIGINAL DISPLACEMENT
0880- B1 3A     2120          LDA (MON.PC),Y
0882- 10 02     2130          BPL .1        POSITIVE DISPLACEMENT
0884- C6 3B     2140          DEC MON.PC+1 DECREMENT HI-BYTE IF NEGATIVE
0886- 20 8E 09  2150 .1       JSR ADD.A.TO.PC
0889- 4C E2 08  2160          JMP UPDATE.PC
                2170 *--------------------------------
088C- 20 97 09  2180 X.BRK   JSR DISPLAY.REGISTERS
088F- 4C 88 09  2190 RTRN.JMP  JMP RETURN
                2200 *--------------------------------
0892- BA        2210 X.RTI    TSX
0893- E0 FD     2220          CPX #$FD
0895- B0 F8     2230          BCS RTRN.JMP
0897- 68        2240          PLA           SIMULATE RTI BY GETTING
0898- 8D 3D 0A  2250          STA SAVE.P    STATUS FROM STACK
089B- BA        2260 X.RTS    TSX
089C- E0 FE     2270          CPX #$FE
089E- B0 EF     2280          BCS RTRN.JMP
08A0- 68        2290          PLA           SIMULATE RTS BY GETTING
08A1- 85 3A     2300          STA MON.PC    PC FROM STACK
08A3- 68        2310          PLA
08A4- 85 3B     2320          STA MON.PC+1
08A6- BA        2330          TSX
08A7- 8E 3C 0A  2340          STX SAVE.S
08AA- 4C E2 08  2350          JMP UPDATE.PC
                2360 *--------------------------------
08AD- 18        2370 X.JSR    CLC           UPDATE PC AND PUSH ON STACK
08AE- A5 3A     2380          LDA MON.PC
08B0- 69 02     2390          ADC #2
08B2- A8        2400          TAY           SAVE LO-BYTE FOR NOW
08B3- A5 3B     2410          LDA MON.PC+1
08B5- 69 00     2420          ADC #0
08B7- 48        2430          PHA           PUSH HI-BYTE
08B8- 98        2440          TYA
08B9- 48        2450          PHA           PUSH LO-BYTE
08BA- BA        2460          TSX
08BB- 8E 3C 0A  2470          STX SAVE.S
08BE- 20 EB 08  2480 X.JMP    JSR GET.NEW.PC
08C1- 4C 28 08  2490          JMP TRACE.LOOP
08C4- 20 EB 08  2500 X.JMPI   JSR GET.NEW.PC
08C7- A0 00     2510          LDY #0
08C9- 20 ED 08  2520          JSR GET.NEW.PC.0
08CC- 4C 28 08  2530          JMP TRACE.LOOP
```

```
                *------------------------------
                DID.NOT.BRANCH
08CF- 8D 40 0A         STA SAVE.A     SAVE ALL REGISTERS
08D2- 8E 3F 0A         STX SAVE.X
08D5- 8C 3E 0A         STY SAVE.Y
08D8- 08               PHP
08D9- 68               PLA
08DA- 8D 3D 0A         STA SAVE.P
08DD- BA               TSX
08DE- 8E 3C 0A         STX SAVE.S
08E1- D8               CLD
                UPDATE.PC
08E2- 38               SEC            0=1, 1=2, 2=3
08E3- A5 2F            LDA MON.LENGTH
08E5- 20 8E 09         JSR ADD.A.TO.PC
08E8- 4C 28 08         JMP TRACE.LOOP
                *------------------------------
                GET.NEW.PC
08EB- A0 01            LDY #1         GET NEW PC FROM INSTRUCTION
                GET.NEW.PC.0
08ED- B1 3A            LDA (MON.PC),Y
08EF- AA               TAX            SAVE LO-BYTE FOR NOW
08F0- C8               INY
08F1- B1 3A            LDA (MON.PC),Y
08F3- 85 3B            STA MON.PC+1 NEW HI-BYTE
08F5- 86 3A            STX MON.PC     NEW LO-BYTE
08F7- 60               RTS
                *------------------------------
                WAIT.ON.KEYBOARD
08F8- A9 16            LDA #22        LINE 23
08FA- 85 25            STA MON.CV
08FC- A9 1A            LDA #26        COLUMN 27
08FE- 85 24            STA MON.CH
0900- 20 22 FC         JSR MON.VTAB
0903- 20 0C FD         JSR MON.RDKEY
0906- C9 8D            CMP #$8D
0908- F0 7E            BEQ RETURN
090A- C9 A0            CMP #$A0
090C- D0 01            BNE .1         REGISTER NAME
090E- 60               RTS
090F- A0 04     .1     LDY #4
0911- D9 7D 09  .2     CMP REG.NAMES,Y
0914- F0 05            BEQ .3
0916- 88               DEY
0917- 10 F8            BPL .2
0919- 30 DD            BMI WAIT.ON.KEYBOARD
091B- 8C 82 09  .3     STY REG.INDEX
091E- 20 80 FE  .4     JSR MON.SETINV
0921- A9 16            LDA #22
0923- 85 25            STA MON.CV
0925- 20 22 FC         JSR MON.VTAB
0928- AC 82 09         LDY REG.INDEX
092B- B9 83 09         LDA REG.CH,Y
092E- 85 24            STA MON.CH
0930- B9 3C 0A         LDA SAVE.AREA,Y
0933- 20 DA FD         JSR MON.PRBYTE
0936- 20 84 FE         JSR MON.SETNORM
0939- AD 00 C0  .5     LDA KEYBOARD
093C- 10 FB            BPL .5
093E- 8D 10 C0         STA STROBE
0941- C9 A0            CMP #$A0       BLANK?
0943- F0 22            BEQ .8         YES
0945- C9 8D            CMP #$8D       RETURN?
0947- F0 28            BEQ .9         YES
0949- 49 B0            EOR #$B0
094B- C9 0A            CMP #10
094D- 90 06            BCC .6         DIGIT
094F- 69 88            ADC #$88
0951- C9 FA            CMP #$FA
0953- 90 E4            BCC .5         NOT DIGIT, SO IGNORE
0955- A0 03     .6     LDY #3
0957- 0A               ASL
0958- 0A               ASL
0959- 0A               ASL
095A- 0A               ASL
095B- AE 82 09         LDX REG.INDEX
```

```
095E- 0A        3300 .7       ASL
095F- 3E 3C 0A  3310          ROL SAVE.AREA,X
0962- 88        3320          DEY
0963- 10 F9     3330          BPL .7
0965- 30 B7     3340          BMI .4         ...ALWAYS
0967- AC 82 09  3350 .8       LDY REG.INDEX
096A- 88        3360          DEY
096B- 10 AE     3370          BPL .3
096D- A0 04     3380          LDY #4
096F- D0 AA     3390          BNE .3         ...ALWAYS
0971- A9 17     3400 .9       LDA #23
0973- 85 23     3410          STA MON.WNDBTM
0975- 20 70 FC  3420          JSR MON.SCROLL
0978- E6 23     3430          INC MON.WNDBTM
097A- 4C F8 08  3440          JMP WAIT.ON.KEYBOARD
097D- D3 D0 D9
0980- D8 C1     3450 REG.NAMES .AS -/SPYXA/
0982-           3460 REG.INDEX .BS 1
0983- 26 23 20
0986- 1D 1A     3470 REG.CH    .DA #38,#35,#32,#29,#26
                3480 *--------------------------------
0988- 20 42 FC  3490 RETURN JSR MON.CLREOP
098B- 4C D0 03  3500        JMP DOS.REENTRY
                3510 *--------------------------------
                3520 *      ADD (A) TO MON.PC
                3530 *--------------------------------
                3540 ADD.A.TO.PC
098E- 65 3A     3550          ADC MON.PC
0990- 85 3A     3560          STA MON.PC
0992- 90 02     3570          BCC .1
0994- E6 3B     3580          INC MON.PC+1
0996- 60        3590 .1       RTS
                3600 *--------------------------------
                3610 *      DISPLAY REGISTERS
                3620 *--------------------------------
                3630 DISPLAY.REGISTERS
0997- A9 1A     3640          LDA #26
0999- 85 24     3650          STA MON.CH
099B- A2 04     3660          LDX #4
099D- D0 05     3670          BNE .2
099F- A9 A0     3680 .1       LDA #$A0
09A1- 20 ED FD  3690          JSR MON.COUT
09A4- BD 3C 0A  3700 .2       LDA SAVE.AREA,X
09A7- 20 DA FD  3710          JSR MON.PRBYTE
09AA- CA        3720          DEX
09AB- 10 F2     3730          BPL .1
09AD- 60        3740          RTS
                3750 *--------------------------------
09AE- 20 3C 53
09B1- 50 43 3E
09B4- 3D 4E 45
09B7- 58 54 20
09BA- 20 3C 52
09BD- 45 54 3E
09C0- 3D 51 55
09C3- 49 54 20
09C6- 20 20 41
09C9- 20 20 58
09CC- 20 20 59
09CF- 20 20 50
09D2- 20 20 53
09D5- 20        3760 BOTTOM.LINE .AS / <SPC>=NEXT  <RET>=QUIT   A  X  Y  P  S /
09D6- 00        3770          .HS 00
                3780 *--------------------------------
                3790 *      PRINT PC AND DASH
                3800 *--------------------------------
                3810 PRINT.PC
09D7- A6 3A     3820          LDX MON.PC
09D9- A4 3B     3830          LDY MON.PC+1
09DB- 4C 99 FD  3840          JMP MON.PRYX3
```

Lines 4440-4480: A test program for you to try STEPping through. Another neat program to trace is at $FCA8 in the monitor (a delay loop).

```
                  3850 *---------------------------------
                  3860 *      DISASSEMBLE NEXT OPCODE
                  3870 *---------------------------------
                  3880 DISASSEMBLE
09DE- 20 D7 09    3890        JSR PRINT.PC
09E1- A0 00       3900        LDY #0
09E3- B1 3A       3910        LDA (MON.PC),Y GET OPCODE
09E5- 20 8E F8    3920        JSR MON.INSDS2
09E8- 48          3930        PHA        SAVE MNEMONIC TABLE INDEX
09E9- B1 3A       3940 .1     LDA (MON.PC),Y
09EB- 20 DA FD    3950        JSR MON.PRBYTE
09EE- A2 01       3960        LDX #1     PRINT ONE BLANK
09F0- 20 4A F9    3970 .2     JSR MON.PRBL2
09F3- C4 2F       3980        CPY MON.LENGTH
09F5- C8          3990        INY
09F6- 90 F1       4000        BCC .1
09F8- A2 03       4010        LDX #3
09FA- C0 03       4020        CPY #3
09FC- 90 F2       4030        BCC .2
09FE- 68          4040        PLA        GET MNEMONIC TABLE INDEX
09FF- A8          4050        TAY
0A00- B9 C0 F9    4060        LDA MNEML,Y
0A03- 85 2C       4070        STA LMNEM
0A05- B9 00 FA    4080        LDA MNEMH,Y
0A08- 85 2D       4090        STA RMNEM
0A0A- A9 00       4100 .3     LDA #0
0A0C- A0 05       4110        LDY #5
0A0E- 06 2D       4120 .4     ASL RMNEM  SHIFT 5 BITS OF CHARACTER INTO A
0A10- 26 2C       4130        ROL LMNEM
0A12- 2A          4140        ROL
0A13- 88          4150        DEY
0A14- D0 F8       4160        BNE .4
0A16- 69 BF       4170        ADC #$BF
0A18- 20 ED FD    4180        JSR MON.COUT
0A1B- CA          4190        DEX
0A1C- D0 EC       4200        BNE .3
0A1E- A9 A0       4210        LDA #$A0    PRINT BLANK
0A20- 20 ED FD    4220        JSR MON.COUT
0A23- 20 0C F9    4230        JSR MON.PRADDR
0A26- 20 9C FC    4240        JSR MON.CLREOL
0A29- 20 8E FD    4250        JSR MON.CROUT
0A2C- A0 27       4260        LDY #39
0A2E- B9 AE 09    4270 .5     LDA BOTTOM.LINE,Y
0A31- 29 3F       4280        AND #$3F
0A33- 99 D0 07    4290        STA BASE.LINE24,Y
0A36- 88          4300        DEY
0A37- 10 F5       4310        BPL .5
0A39- C6 25       4320        DEC MON.CV
0A3B- 60          4330        RTS
                  4340 *---------------------------------
                  4350 SAVE.AREA
0A3C-             4360 SAVE.S .BS 1
0A3D-             4370 SAVE.P .BS 1
0A3E-             4380 SAVE.Y .BS 1
0A3F-             4390 SAVE.X .BS 1
0A40-             4400 SAVE.A .BS 1
                  4410 *---------------------------------
                  4420 *      TEST PROGRAM
                  4430 *---------------------------------
0A41- 20 45 0A    4440 TEST   JSR TEST1
0A44- 00          4450        BRK
0A45- 20 48 0A    4460 TEST1  JSR TEST2
0A48- 20 4B 0A    4470 TEST2  JSR TEST3
0A4B- 60          4480 TEST3  RTS
```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage). All material herein is copyrighted by S-C SOFTWARE, all rights reserved. Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)